2ND EDITION

Reactive Patterns with RxJS and Angular Signals

Elevate your Angular 18 applications with RxJS Observables, subjects, operators, and Angular Signals

LAMIS CHEBBI

Foreword by Aristeidis Bampakos, Angular Google Developer Expert (GDE)

Reactive Patterns with RxJS and Angular Signals

Elevate your Angular 18 applications with RxJS Observables, subjects, operators, and Angular Signals

Lamis Chebbi

‹packt›

Reactive Patterns with RxJS and Angular Signals

Group Product Manager: Kaustubh Manglurkar

Publishing Product Manager: Vaideeshwari Muralikrishnan

Senior Editor: Hayden Edwards

Technical Editor: Simran Ali

Copy Editor: Safis Editing

Project Coordinator: Shagun Saini

Proofreader: Safis Editing

Indexer: Rekha Nair

Production Designer: Jyoti Kadam

Marketing Coordinator: Anamika Singh and Nivedita Pandey

First edition published: April 2022

Second edition published: July 2024

Production reference: 1290524

Published by Packt Publishing Ltd

Grosvenor House

11 St Paul's Square

Birmingham

B3 1RB

ISBN 978-1-83508-770-1

www.packtpub.com

To my father, who instilled in me diligence, perseverance, and a good work ethic. Thank you for always being there to support me and lift me up.

To my mother, who taught me selflessness and doing things with love. Thank you for your enduring encouragement during the writing of this book.

To my brother and my sisters, for their continuous support.

– Lamis Chebbi

Foreword

RxJS is a powerful JavaScript library that enables developers to build reactive and event-based web applications. The Angular framework uses this library to manage asynchronous operations, such as HTTP communication and user interaction with web forms and routing.

Angular Signals, a cutting-edge API, introduces fine-grained reactivity in Angular applications. This synchronous reactive pattern boosts performance and intelligently tracks the application state, optimizing component rendering and enhancing the overall user experience.

Reactive Patterns with RxJS and Angular Signals is a book that embraces both worlds, combining best practices from each tool to help you build performant and reactive Angular applications.

Lamis uses a simple yet insightful approach to RxJS and Signals, one which not only allows you to gain a deeper understanding of all the available reactive patterns in Angular, but also one that helps you build a complete application that encompasses all the latest features of the Angular framework.

Aristeidis Bampakos

Angular Google Developer Expert (GDE)

Contributors

About the author

Lamis Chebbi is a Google Developer Expert for Angular and is the author of the first edition of this book, titled *Reactive Patterns with RxJS for Angular*. She is an enthusiastic software engineer with a strong passion for the modern web, the founder of Angular Tunisia, a member of the WWCode community, a speaker, a content creator, and a trainer.

She has been interested in Angular and RxJS for the past few years and loves to share her knowledge about Angular by participating in workshops and organizing training sessions. Empowering women and students is one of her highest priorities.

Besides Angular and the web, Lamis loves music, traveling, chromotherapy, and volunteering.

Last but not least, she's a forever student.

I want to thank all the people who believed in me, supported me, and inspired me throughout this journey.

About the reviewers

Aleksandr Guzenko is a respected software engineer known for his extensive expertise in both frontend and backend development. With over eight years of experience in the field, Aleksandr has made significant contributions to the software engineering community. His deep knowledge and practical skills are not only evident in his professional work but also in his active involvement as a judge in numerous hackathons.

Furthermore, Aleksandr is a sought-after speaker at conferences, where he shares his insights and experiences, particularly focusing on software architecture. He is also an author, of articles that delve into various aspects of software engineering.

Ishu Mishra is an IT specialist, working on frontend tech stacks for over 6 years. He began working at a start-up organization called Sparx IT Solutions, where he gained knowledge of Angular, and has worked for several companies since then.

Ishu previously worked for the Japanese multinational NEC Corporation, and presently, Ishu is employed in Bangalore by Morgan Stanley.

Matheus Rian is a frontend developer and speaker, who is passionate about technology and education. He started programming in high school and hasn't stopped learning since, developing his skills in technologies such as Angular and React.

Furthermore, Matheus is a multiplier, seeking to disseminate knowledge and generate a positive impact within communities. As well as working in the frontend market, he gives lectures at events and contributes articles on Medium and `dev.to`.

Arthur Lannelucq is a passionate frontend developer specializing in Angular and RxJS. He is a strong advocate for reactive architectures, believing in their power to build responsive and scalable web applications.

He has gained solid experience working on various projects for large companies and start-ups. Eager to share his knowledge, he also runs a YouTube channel where he provides tutorials and practical advice on Angular and frontend development.

Table of Contents

Part 2: A Trip into Reactive Patterns

3

4

5

Combining Streams 71

6

Transforming Streams 87

7

Sharing Data between Angular Components 111

Part 3: The Power of Angular Signals

8

Mastering Reactivity with Angular Signals 123

Part 4: Multicasting Adventures

9

Demystifying Multicasting 151

10

Boosting Performance with Reactive Caching 165

11

Performing Bulk Operations 177

12

Processing Real-Time Updates 189

Part 5: Final Touches

13

Preface

Embarking on the journey from imperative to reactive programming is a significant shift and one that I have experienced firsthand. As I navigated this transition, I found myself drawn to the world of reactive patterns and the transformative power they held. It was a journey filled with discovery, comparison, and a strong determination to understand this new way of thinking.

Inspired by my own experiences, I've crafted this book to serve as a guide through the realms of reactive patterns within Angular applications. I believe that the reactive mindset is gradually achieved by comparing the reactive way to the imperative way, in order to distinguish the difference and benefits. Within these pages, you'll discover how embracing reactive patterns can greatly enhance the way you manage data, write code, and react to user changes. From improving efficiency to creating cleaner, more manageable code bases, the benefits are vast and practical.

So, without further ado, let's embark on this journey together and unlock the potential of reactive programming.

Who this book is for

If you're a developer working with Angular and RxJS, this book is tailor-made for you. Designed for individuals at a beginner level in both Angular and RxJS, this book will guide you toward becoming an experienced developer while also benefitting those who wish to harness the potential of RxJS and leverage the reactive paradigm within their Angular applications.

What this book covers

In *Chapter 1*, *Diving into the Reactive Paradigm*, you will learn the fundamentals of reactive programming.

In *Chapter 2*, *Walking through Our Application*, you will learn the architecture and requirements of the recipe application that we will be building through the book.

In *Chapter 3*, *Fetching Data as Streams*, you will learn the reactive pattern for fetching data so that we can reactively retrieve a list of recipes in our recipe app.

In *Chapter 4*, *Handling Errors Reactively*, you will learn the different error strategies and the reactive patterns for handling errors.

In *Chapter 5*, *Combining Streams*, you will learn the reactive pattern for combining streams and use it to implement a filter functionality in our recipe app, while also discovering the common pitfalls and sharing best practices for optimal implementation.

In *Chapter 6*, *Transforming Streams*, you will learn the reactive pattern for transforming streams and use it to implement autosave and autocomplete features in our recipe app.

In *Chapter 7*, *Sharing Data between Angular Components*, you will learn the reactive pattern to share data between components and use it to share the selected recipe in our recipe app.

In *Chapter 8*, *Mastering Reactivity with Angular Signals*, you will deep-dive into Angular signals, learning different reactive patterns based on Angular Signals, and how to unleash the power of RxJS and Signals together. You will also discover the latest Angular Signals improvements.

In *Chapter 9*, *Demystifying Multicasting*, you will learn the essentials of multicasting and the different multicasting concepts and operators offered by RxJS, such as Subjects, Behavior Subjects, and Replay Subjects.

In *Chapter 10*, *Boosting Performance with Reactive Caching*, you will learn the reactive pattern to cache streams and implement a caching mechanism in our recipe app, based on the latest RxJS features.

In *Chapter 11*, *Performing Bulk Operations*, you will learn the reactive pattern to perform bulk operations and implement a multiple asynchronous file upload in our recipe app.

In *Chapter 12*, *Processing Real-Time Updates*, you will explore the reactive patterns to consume real-time updates and display newly created recipes instantly in our recipe app.

In *Chapter 13*, *Testing RxJS Observables*, you will learn the different strategies to test reactive patterns and practice testing the API responses in our recipe app.

To get the most out of this book

This book assumes some familiarity with Angular, basic RxJS, TypeScript, and a foundational knowledge of functional programming. All code examples have been tested using Angular 17 and 18 on on the Windows OS. However, they should work with future version releases too.

Software/hardware covered in the book	**Operating system requirements**
Angular 17 and above	Windows, macOS, or Linux
TypeScript 5.4.2	Windows, macOS, or Linux
RxJS 7.8.1	Windows, macOS, or Linux
PrimeNG 17.10.0	Windows, macOS, or Linux
Bootstrap 5.0.0	Windows, macOS, or Linux

Make sure you follow the prerequisites found here: `https://angular.dev/tools/cli/setup-local`. The prerequisites include the environment setup and the technologies needed in order to install and use Angular.

We also use the Bootstrap library to manage the application's responsiveness, the PrimeNG library for its rich components, and, of course, RxJS as the reactive library.

Plus, there is a ready-for-use backend server in the GitHub repository that we will only reference in our application.

If you are using the digital version of this book, we advise you to type the code yourself or access the code from the book's GitHub repository (a link is available in the next section). Doing so will help you avoid any potential errors related to the copying and pasting of code.

Download the example code files

You can download the example code files for this book from GitHub at `https://github.com/PacktPublishing/Reactive-Patterns-with-RxJS-and-Angular-Signals-Second-Edition`. If there's an update to the code, it will be updated in the GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Conventions used

There are a number of text conventions used throughout this book.

`Code in text`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "In the following code snippet, we have an example of an Angular service that injects the `HttpClient` service and fetches data from the server using the `HttpClient.get()` method."

A block of code is set as follows:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable} from 'rxjs';
```

Any command-line input or output is written as follows:

```
//console output
Full Name: John Doe
```

Bold: Indicates a new term, an important word, or words that you see on screen. For instance, words in menus or dialog boxes appear in **bold**. Here is an example: "Users can create a new recipe by clicking on the **New Recipe** menu item located at the top right of the page."

Tips or important notes
Appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, email us at `customercare@packtpub.com` and mention the book title in the subject of your message.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata` and fill in the form.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packtpub.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Share Your Thoughts

Once you've read *Reactive Patterns with RxJS and Angular Signals*, we'd love to hear your thoughts! Please `click here to go straight to the Amazon review page` for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below

`https://download.packt.com/free-ebook/9781835087701`

2. Submit your proof of purchase
3. That's it! We'll send your free PDF and other benefits to your email directly

Part 1: An Introduction to the Reactive World

Embark on a journey into the world of reactive programming with Angular!

In this part, you will understand the fundamentals of the reactive paradigm and its application in Angular, gaining insight into why it's essential to leverage this approach. Then, we will introduce the recipe application that we are going to progressively build as we go through the book.

This part includes the following chapters:

- *Chapter 1*, *Diving into the Reactive Paradigm*
- *Chapter 2*, *Walking through Our Application*

1
Diving into the Reactive Paradigm

Reactive patterns are reusable solutions to a commonly occurring problem using reactive programming. Behind all these patterns is a new way of thinking, a new architecture, new coding styles, and new tools. That's what this entire book is based on – useful reactive patterns in Angular applications.

Now, I know you are impatient to write your first reactive pattern in Angular, but before doing so, and in order to help you take full advantage of all the RxJS patterns and leverage the reactive paradigm, we will start by explaining in detail all the fundamentals and preparing the groundwork for the following chapters.

Let's start with a basic understanding of the reactive paradigm, its advantages, and the problems it solves. Best of all, let's put a reactive mindset on and start thinking reactively. We will begin by highlighting the pillars and the advantages of the reactive paradigm. Then, we will explain the marble diagram and why it is useful. Finally, we will highlight the use of RxJS in Angular.

Giving an insight into the fundamentals of the reactive paradigm is incredibly important. This will ensure you get the basics right, help you understand the usefulness of the reactive approach, and consequently help you determine which situation is best to use it in.

In this chapter, we're going to cover the following topics:

- Exploring the pillars of reactive programming
- Learning the marble diagram (our secret weapon)
- Highlighting the use of RxJS in Angular

Technical requirements

This chapter does not require any environment setup or installation steps.

All the code snippets in this chapter are just examples to illustrate the concept, so you will not need the code repository to follow along. However, if you're interested, the code for the book can be found at `https://github.com/PacktPublishing/Reactive-Patterns-with-RxJS-and-Angular-Signals-Second-Edition`.

This book assumes that you have a basic understanding of Angular and RxJS.

> **Note**
>
> This book uses the new Angular documentation site, `angular.dev`. The previous documentation site, `angular.io`, will soon be deprecated. Stay connected with the latest updates and resources by accessing the documentation through this link.

Exploring the pillars of reactive programming

Reactive programming is among the major programming paradigms used by developers worldwide. Every programming paradigm solves some problems and has its own advantages. By definition, reactive programming is programming with asynchronous data streams and is based on observer patterns. So, let's talk about these pillars of reactive programming!

Data streams

Data streams are the spine of reactive programming. Everything that may change or happen over time (you don't know when exactly) is represented as asynchronous streams such as events, notifications, and messages. Reactive programming is about reacting to changes as soon as they are emitted!

An excellent example of data streams is UI events. Let's suppose that we have an HTML button and we want to execute an action whenever a user clicks on it. Here, we can think of the click event as a stream:

```
//HTML code
<button id='save'>Save</button>

//JS code
const saveElement = document.getElementById('save');
saveElement.addEventListener('click', processClick);

function processClick(event) {
  console.log('Hi');
}
```

As implemented in the preceding code snippet, in order to react to the click event, we register an `EventListener` event. Then, every time a click occurs, the `processClick` method is called to execute a side effect. In our case, we are just logging `Hi` in the console.

As you might have gathered, to be able to react when something happens and execute a side effect, you should listen to the streams to become notified. To get closer to reactive terminology, instead of *listen*, we can say *observe*. This leads us to the *observer* design pattern, which is at the heart of reactive programming.

Observer patterns

The **observer pattern** is based on two main roles – a publisher and a subscriber:

- A **publisher** maintains a list of subscribers and notifies them or propagates a change every time there is an update
- On the other hand, a **subscriber** performs an update or executes a side effect every time they receive a notification from the publisher

The observer pattern is illustrated here:

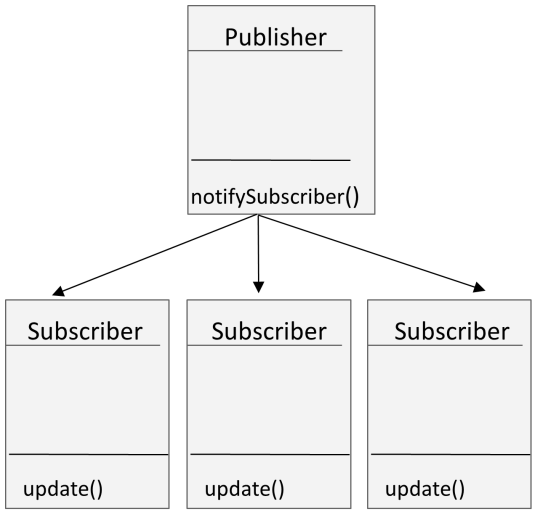

Figure 1.1 – The observer pattern

To get notified about the updates, you need to subscribe to the publisher. A real-world analogy would be a newsletter; you don't get any emails from a specific newsletter if you don't subscribe to it.

This leads us to the building blocks of RxJS, which include the following:

- **Observables**: These are a representation of the asynchronous data streams that notify the observers of any change
- **Observers**: These are consumers of the data streams emitted by Observables

RxJS combines the observer pattern with the iterator pattern and functional programming to process and handle asynchronous events. This was a reminder of reactive programming fundamentals, and it is crucial to know when to put a reactive implementation in place and when to avoid it.

In general, whenever you have to handle asynchronous tasks in your Angular application, always think of RxJS. The main advantages of RxJS over other asynchronous APIs are as follows:

- RxJS makes dealing with event-based programs, asynchronous data calls, and callbacks an easy task.
- Observables guarantee consistency. They emit multiple values over time so that you can consume continuous data streams.
- Observables are lazy; they are not executed until you subscribe to them. This helps with writing declarative code that is clean, efficient, and easy to understand and maintain.
- Observables can be canceled, completed, and retrieved at any moment. This makes a lot of sense in many real-world scenarios.
- RxJS provides many operators with a functional style to manipulate collections and optimize side effects.
- Observables push errors to the subscribers and provide a clean way to handle errors.
- RxJS allows you to write clean and efficient code to handle asynchronous data in your application.

Now that we have given some insight into the reactive programming pillars and detailed the major advantages of RxJS, let's explore the marble diagram, which is very handy for understanding and visualizing the Observable execution.

Learning about the marble diagram (our secret weapon)

RxJS ships with more than one hundred **operators** – these are among the building blocks of RxJS, useful for manipulating streams. All the reactive patterns that will be detailed later in this book are based on operators, and when it comes to explaining operators, it is better to refer to a visual representation – that's where marble diagrams come in!

Marble diagrams are visual representations of the operator's execution, which will be used in all chapters to understand the behavior of RxJS operators. At first, it might seem daunting, but it is delightfully simple. You only have to understand the anatomy of the diagram and then you'll be good at reading and translating it.

Marble diagrams represent the execution of an operator, so every diagram will include the following:

- **Input Observable(s)**: Represents one or many Observables given as input to the operator
- **Operator**: Represents the operator to be executed with its parameters
- **Output Observable**: Represents the Observable produced after the operator's execution

We can see the execution illustrated here:

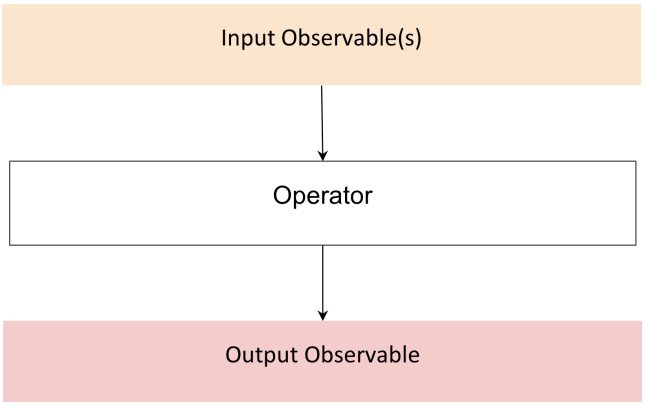

Figure 1.2 – The operator execution

Now, let's zoom in on the representation of the input/output Observables:

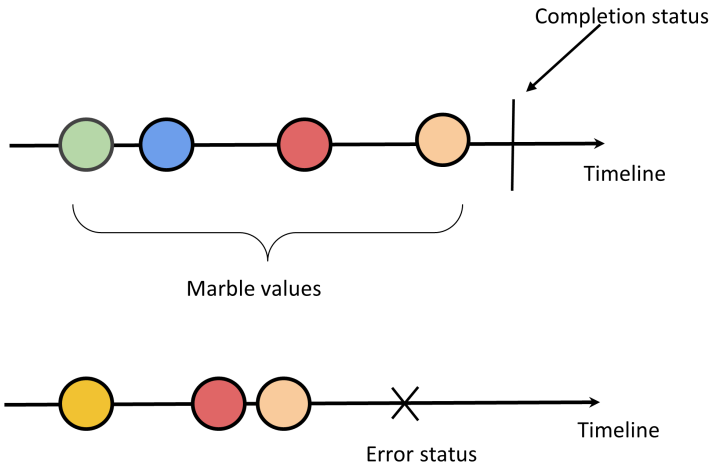

Figure 1.3 – The marble diagram elements